Theorem 3.3.2.1:

$\forall dsgn.\ (\text{well_formed_design_sigs}\ dsgn) \supset (\text{get_ports}\ dsgn \subseteq \text{get_sigs_design}\ dsgn)$

The proof was performed in the theorem prover by induction on the structure of designs. This proof, because of its mechanical nature, is not given here; but the argument is fairly straightforward. The interested reader is referred to Appendix A for the actual proof script. The base case follows from an expansion of the definition of well_formed_design_sigs. Once done, one observes that the ports and declarations of the component must be the same as the signals of its concurrent statements (derived by get_sigs_cs). This of course means that the ports are a subset of the total signals. The induction step is also a based on a straightforward expansion of definitions. Once those of well_formed_design_sigs, get_ports and get_sigs_design have been applied to composite designs one arrives at an instance of the induction hypothesis.

3.3.3 Overall Well-Formedness

It is now possible to make an overall definition about the syntactic well-formedness of any Femto-VHDL design. Simply stated, the signals of the design must exhibit the appropriate relationships, and the concurrent statements must not contain multiply driven signals. These requirements are formally defined in the function well_formed_design:

well_formed_design $design \equiv$
$(\text{well_formed_design_sigs}\ design) \wedge (\text{wfF_csss}\ (\text{dflat}\ design))$

A simple fact arising from the definition of well_formed_design is:

Theorem 3.3.3.1:

$\forall design.\ \text{well_formed_design}\ design \supset \text{well_formed_design_sigs}\ design$

The intuition for the proof is to assume well_formed_design $design$, expand the definition and take the first conjunct.

Well-Formedness of Design Signals

In order to ensure the well-formedness of the signals in a Femto-VHDL program, the various constraints on signals mentioned earlier must be met. To reiterate, they were:

- Only declared signals may be used in the program text.
- Directionality of assignments must be preserved.

The function well_formed_design_sigs uses the just defined functions to express these goals for the well-formedness of the signals in a Femto-VHDL design.

well_formed_design_sigs $design \equiv$
  if $design =$ arch $ent\ arch\ ports\ decls\ cs$ then
    let $ip =$ fst $ports$ and $op =$ fst (snd $ports$) and $op =$ snd (snd $ports$)
    and $(isigcs, osigcs) =$ get_sigs_cs cs and $d =$ get_sigs_decls $decls$ in
    let $p = (ip \cup op \cup iop)$ and $sigcs = (isigcs \cup osigcs)$ in
    let $pId = (p \cap d = \{\})$ and $oIi = (op \cap isigcs = \{\})$ and $iIo = (ip \cap osigcs = \{\})$ in
      $(sigcs = p \cup d) \wedge pId \wedge oIi \wedge iIo$
  else
    let (dpar $d'\ d''$) $= design$ in
    let $(P', S') =$ well_formed_design_sigs d'
    and $(P'', S'') =$ well_formed_design_sigs d'' in
      $(P' \wedge P'')$

In the case of stand-alone components, *ip* are the `in` ports, *op* are the `out` ports and *iop* are the `inout` ports. *isigs* represents the signals that are used as inputs in the concurrent statements of the component, whilst *osigs* are those used as outputs. *d* are the locally declared signals. The variable *p* holds all the ports, and *sigcs* contains all the signals of the concurrent statements. *pId* states that the ports and locally declared signals must be disjoint, *oIi* says that the output ports must not be used as inputs, and *iIo* ensures that input ports are not being used as outputs. Conjoining these last three with the statement that the ports of the component are the same as the signals used by the concurrent statements ensures that the all the signals of that component are well-formed.

A useful consequence of the definition of well_formed_design_sigs when reasoning about the dynamic semantics of Femto-VHDL in Chapter 4, will be the knowledge that if the signals in a design display the proper interrelationships, then the ports of the design are a subset of all the signals in that design.

Finally, it is necessary to traverse an entire sea of concurrent processes gathering the same information. The function get_sigs_cs does this, and returns a pair of sets of the same form as those created by get_sigs_ss.

get_sigs_cs $cs \equiv$
  if $cs = (sl : ss)$ then get_sigs_ss ss
  else let $(cs' \,||\, cs'') = cs$ in
    let $(is', os') =$ get_sigs_cs cs' and $(is'', os'') =$ get_sigs_cs cs'' in
      $(is' \cup is'', os' \cup os'')$

As an example of the behaviour of this suite of functions, take the Femto-VHDL processes:

$\{x\} : a \overset{\text{inert}}{:=} (\text{not } x, 2) \,||\, \{y'\} : b \overset{\text{trans}}{:=} (y' \text{ and } y'', 0) \,||\, \{z\} : c \overset{\text{trans}}{:=} (z, 0)$